Alim Nizari, Christopher Kassner, John Ye, Thomas Chiu

CMPT 276 Phase 2 Report

Our overall approach to implementing the game was quite different from the design and structure originally envisioned during the first phase of the game's development. During the design phase, we originally sought to make a distinction between animate and inanimate classes, both of which were aggregates of the cells on the dungeon's game board. Also, we planned on having the game's user UI consisting of a start menu, pause menu, volume slider and various language options. However, during the actual implementation of the game, we found that the distinction between the animate and inanimate classes as well as the cell class were unnecessary and much of the game's UI was not implemented due to time constraints and prioritization of tasks.

During the implementation phase, our first priority was to realize the game's board and then figure out the logic behind player movement. Then, we slowly began to build and expand our game through a bottom-up approach revolving around our implementation of the game's board and the logic behind player movement. Some of our earlier tasks included moving the player around the board and checking for walls, as well as finding appropriate sprites for the walls, floor, player, enemies, items and punishments and rendering graphics for the game. Some of our later tasks included implementing enemy movement and displaying the game's score and timer.

Regarding the development process, various adjustments and modifications to the initial design proved necessary. Our initial UML diagram featured a diverse class hierarchy including a board class, with sub-classes for animate objects such as the player and moving enemies, and inanimate objects such as rewards and punishments. However, in our actual implementation of the game, we wanted to create a model that would give us a quick visual representation of how all of our pieces would interact with one another. This led us to creating a basic 2D integer array to represent our board, with different integers representing different classes. Building off this model, we were able to incrementally implement various other features such as the graphics and movement logic for our main character. At this point, our project only consisted of three classes - Board, Game, and Main. Since a large portion of our core functionality was achievable purely through the use of our 2D integer array, we decided to stick with it instead of refactoring into a cell array. Many of the subclasses seemed redundant, however with our largely classless implementation we faced issues in completing the last few necessary functionalities which mainly revolved around the movement of the player and enemies. Thus, classes for our player and enemies were created, and much of our naive implementation in our Board class had to be ported over to our Game class with the introduction of our game loop. Although various modifications were made in terms of the interactions between different classes and overall structure, our game did consist of separate classes for the player, enemy, rewards, punishments and board as per the original UML diagram.

Concerning our original use-case templates, there were no adjustments or modifications to our use-cases of high priority. The underlying logic of our game regarding collecting rewards, moving to the end cell, and avoiding enemies were all implemented in our code as originally designed. However, our use-cases of low priority such as adjusting volume, changing the language or opening a pause menu were not implemented into the game, due to time constraints and given that core functionality was our primary concern.

As for our description of the project during the design phase, there were some adjustments. We had originally planned on having two types of moving enemies as well as three types of bonus rewards, including gold coins, invisibility mushrooms and speed potions. However, during the implementation phase of the game only bats were added as moving enemies, and only gold coins were added as bonus rewards, similarly due to time constraints. The game's general theme and atmosphere remain unchanged from our description in the design phase.

In terms of team management and division of roles, each team member participated in the design and coding of the game. Instead of assigning specific roles to individual team members, we initially spent some time researching Java and getting familiar with the syntax, as none of our team members had any prior experience coding in Java. Next, a team member implemented the basic logic behind keeping track of player movement in a 2D integer array. Then, once the game's basic logic was implemented, each team member contributed to the project by slowly adding the various necessary features one-by-one, through a bottom-up approach. During this process, the game slowly progressed as team members contributed what they felt most comfortable with and capable of doing to the code, while continually informing other team members of their progress online in order to keep our efforts synchronized. Some team members dealt more with the game's logic and movement, while other team members dealt more with graphics. In contrast to our initial plan of separating roles by creating the sub classes of animate and inanimate objects, it ended up being more of an informal division between front end and back end coding. A more formal approach was adopted towards the end of the coding process, with each team member explicitly stating what they planned on implementing, mainly to deal with time constraints and in order to avoid having two team members working on exactly the same thing. Although our team's approach was largely informal and lacking structure, each team member was extremely supportive, communicative and willing to help other team members with any issues arising during the game's implementation.

During the game's implementation, no external libraries were used. However, we did use two packages. Java AWT and Java Swing were used for creating user interfaces and dealing with graphics and images, such as displaying the score and timer. Specifically, classes such as Color, Font, Graphics and Rectangle in Java AWT were used, as well as the JFrame class in Java Swing, which is simply an extended version of the Frame class in Java AWT.

There are many measures we took to enhance our code, with one important measure being the division of classes. During a large part of the implementation phase, the entire game was contained in three classes - Board, Game and Main. As we continued developing our game, we created many more classes such as ImageData, Key, Punishment, Wall, Enemy and Player, splitting much of the game's functionality. This led to a much cleaner and more readable version of our game. Another measure we took was related to the representation of walls, items, enemies, rewards and so on in our 2D array of integers. Originally, we had a function to draw the game's board with dozens of lines of code specifically dedicated to generating inner walls, items, etc. However, we eventually decided to substitute this function with a text file, with manually entered values for our representations. Again, this led to a cleaner version of our game.

Finally, there were various challenges facing our team during the implementation phase. One of the biggest challenges was simply team coordination. Each team member had a busy schedule with different availabilities, so it was difficult to find a time for the entire team to meet. Since we weren't able to meet in person very often, our team was forced to communicate mainly online, making it more difficult to communicate any ideas or issues that arose during the coding process. However, as previously mentioned, much of this problem was solved by constantly updating the team on individual progress as well as team voice chats on Discord. Another challenge unrelated to the coding process itself, was troubleshooting errors in Eclipse and IntelliJ. There were times when a team member would push the latest changes to the code, only for other members to pull the code and not be able to run the code due to miscellaneous errors. Since none of the team members had any experience using either Eclipse or IntelliJ, this resulted in many hours of searching around the internet for solutions to any errors or issues, as well as the team resorting to alternate methods to work around these issues. For example, a team member would occasionally create a .zip file of the project folder and send it to the other team members, so that they could extract the files and import it into Eclipse or IntelliJ, fixing any errors. A coding related challenge was implementing some of the more complicated functionality into our game, such as having the enemies correctly chase the player according to the player's movement. Otherwise, most of the coding involved in the creation of our game presented less of a challenge than dealing with various IDEs and team coordination.

In conclusion, although our team faced various challenges and difficulties during the implementation phase, the project served as a valuable learning experience not only in terms of Java or coding, but moreso in terms of working as a team.